

Automated Statistics Collection in Action

M. Kandil² A. Lerner¹ V. Markl¹ I. Popivanov² V. Raman¹ D. Zilio²

¹IBM Almaden Research Center
650 Harry Road
San Jose, CA
USA

²IBM Toronto Development Laboratory
8200 Warden Avenue
Markham, ON
Canada

ABSTRACT

If presented with inaccurate statistics, even the most sophisticated query optimizers make mistakes. They may wrongly estimate the output cardinality of a certain operation and thus make sub-optimal plan choices based on that cardinality. Maintaining accurate statistics is hard, both because each table may need a specifically parameterized set of statistics and because statistics get outdated as the database changes. Automated Statistic Collection (ASC) is a new component in IBM DB2 UDB that, without any DBA intervention, observes and analyzes the effects of faulty statistics and, in response, it triggers actions that continuously repair the latter. In this demonstration, we will show how ASC works to alleviate the DBA from the task of maintaining fresh, accurate statistics in several challenging scenarios. ASC is able to reconfigure the statistics collection parameters (e.g. number of frequent values for a column, or correlations between certain column pairs) on a per-table basis. ASC can also detect and guard against outdated statistics caused by high updates/inserts/deletes rates in volatile, dynamic databases. We will also show how ASC works from the inside: from how cardinality mis-estimations are introduced in different kind of operators, to how this error is propagated to later operations in the plan, to how this influences plan choices inside the optimizer.

1. INTRODUCTION

When considering different alternative plans, cost-based optimizers will look for factors about the underlying tables so to decide which plan would perform better. For instance, when joining two distinct sources of data, factors that may make a particular join algorithm stand out are: the relative size of the data source with respect to the available memory, how likely every pair of possible data elements would match (join selectivity), just to cite a few. The database literature broadly refers to information that allows to evaluate such factors as “*the statistics.*”

DB2 UDB stores relevant statistics in its catalog, as de-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA
Copyright 2005 ACM 1-59593-060-4/05/06 ...\$5.00.

Table 1: DB2 Statistics Information

tables	number of rows in a table
columns	number of distinct values for a column
indexes	number of distinct keys, clustering of the table with respect to the index, physical properties of the index
coldist	number of quantiles and frequent values of a column
colgroups	distinct number of values for a group of columns

picted by Table 1. This information is updated by a tool called RUNSTATS, which takes a statistical snapshot of a designated table at one point in time. Other database managers use similar approaches to maintaining statistics for optimization purposes.

There are mainly two problems about statistics that can induce the optimizer to err. Either they are missing (down-right non-existent, or less specific than DB2’s optimizer could potentially need) or they are outdated (represent a past scenario that is no longer valid).

In the absence of all necessary statistics, the optimizer makes assumptions that may drive it for plans that are sub-optimal. These plans usually present cardinality mis-estimations at run-time, in that the optimizer predicted that a given query operator would output a certain number of rows, and at run-time that estimate turned out to be false.

Consider a scenario where the information about *column groups*, which tells the optimizer that certain column pairs influence on each other’s values, is missing from the statistics. The problem here is that, unless told otherwise, the optimizer assumes that if columns’ values are independent, so are selectivity of predicates applied over them, or that

$$\begin{aligned} &selectivity(col_1 = val_1 \wedge col_2 = val_2) = \\ &selectivity(col_1 = val_1) * selectivity(col_2 = val_2) \end{aligned}$$

If, however, the columns are not independent, the cardinality of the composite predicate has to take that into consideration.

Figure 1 depicts an example of the above case. It shows plan fragments annotated with the estimated and actual cardinalities for three predicates (one line above and two lines below an operators name, respectively). The estimated cardinality for the predicate `make = 'Toyota'`, which is eval-

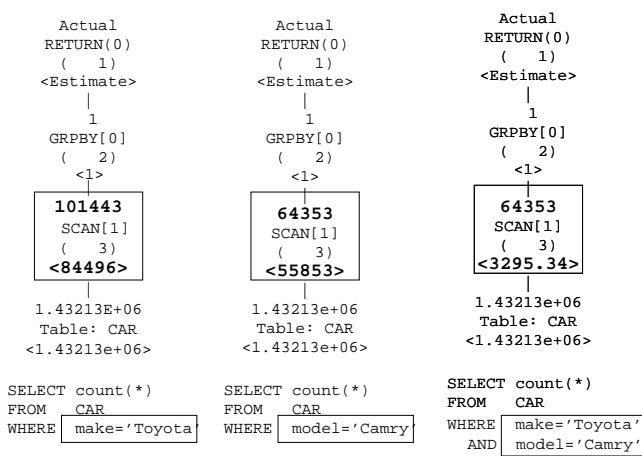


Figure 1: Evidence of independence predicate assumption failure. The one-column predicates cardinalities are within an acceptable margin of error. The combined predicate’s isn’t.

uated at the scan operator, shows an acceptable margin of error (84,496 vs. 101,443). So does the estimation of `model = 'Camry'` (55,853 vs. 64,353). When both predicates are evaluated in a same query, the optimizer applies the independent assumption, generating an estimate that is off by a factor of 20 (3,295 vs. 64,353). This pattern is a strong indicative of a missing *column group* between car’s makes and models.

Even if statistics do exist and are comprehensive enough, they may become outdated in scenarios where databases are volatile. To monitor such effect and to infer when to trigger a refresh of the statistics, one may keep a counter of the number of updates/delete/inserts made over each table (UDI counters).

Automatic Statistics Collection (ASC) is a new component in DB2 UDB that monitors the effects of missing and outdated statistics and recommends appropriate corrective actions. ASC is an autonomic component in that it does not require any DBA tuning once installed.

This demo shows ASC in action. We explore how ASC can help a DBA keep the statistics fresh – and thus the quality of the plans high – under different challenging situations. In one scenario, ASC monitors a select-only database with several faulty and missing statistics while a random workload is being run. In another scenario, ASC is given a dynamic, volatile database in which statistics change, as a mixed workload of queries inquire and change the database. In both scenarios ASC is capable of detecting the faulty statistics and of taking the appropriate corrective measures – while respecting the operational parameters set by the DBA at installation time.

2. ASC’S ARCHITECTURE OVERVIEW

ASC is a component of DB2 UDB that monitors both the query workloads and the data changes they cause – and reacts suggesting changes on the statistics-collection process accordingly. Figure 2 depicts ASC’s architecture. Details about ASC analysis algorithms can be found at [1]. To make this paper self contained, we discuss some details here.

ASC gathers information without interfering with the en-

gine workings. Through its three *monitors*, it collects compile-time information about a query (Plan Monitor), run-time information about the latter (Runtime Monitor), and information about the number of rows updated/deleted/inserted for a table (Activity Monitor).

The Activity Analyzer is the ASC component that reacts to data changes. Once UDI counters are available, the AA uses this information to determine whether data on an active table have changed enough to justify a new RUNSTATS execution. The AA also estimates the degree to which activity on a table may have altered its distribution. The Scheduler uses this information to prioritize tables for statistics collection.

The Query Feedback Analyzer is the ASC component that reacts to runtime cardinality discrepancies. Estimated and actual cardinalities for queries’ operators, which were collected by the Plan and Runtime Monitors, are combined into a Query Feedback Warehouse. Periodically, the QFA mines this warehouse so to find patterns that may indicate incomplete or missing statistics (see Figure 1). The QFA generates modifications in the RUNSTATS profiles (the parameters on how detailed the statistics collection is) according the discrepancies it sees in the warehouse. It also communicates the Scheduler its findings so that this latter can properly prioritize the execution of RUNSTATS.

The Scheduler drives the statistics-collection process. It combines the findings of the QFA and AA and creates a priority list of tables whose statistics should be collected. It invokes RUNSTATS, which will use any recently-updated profile, as a throttled, background process to consume the list of tables. This process respects the window of maintenance parameters the DBA selected at installation time.

3. DEMONSTRATION

We demonstrate ASC using realistic databases and query workloads that test its ability to improve the quality of plans – of queries both inside and outside the workload – by maintaining necessary statistics up-to-date. We use read-only databases with inaccurate/incomplete statistics as well as more volatile databases, where the statistics change due to data changes. The story-line of the demonstration is the following.

1. We start with a previously loaded database about cars, drivers, and accidents. The tables are populated so that several nuances about data distribution are introduced that are not immediately caught by the default parameters of RUNSTATS. We show how a DBA would control the behavior of autonomic background activities by configuring the Scheduler. For instance, the DBA can limit the scope of automated statistics collection to certain tables, or exclude tables from automatic maintenance. The DBA can also specify the maintenance window over which ASC can take its corrective actions. Still, he or she can control whether the Scheduler should invoke the QFA, or the AA, or both, and specify the maximum allowable disk space for the QFW. After this initial parameterization, ASC operates completely unattended.
2. We run a *random* read-only workload, which requires the optimizer to use the available statistics at this point. We show several queries’ annotated plans (such as those of figure 1), where the audience can see how

